

Integrated Cartographic and Programmatic Access to Spatial Data using Object-Oriented Databases in an Online Web GIS

Ionut Iosifescu, Lorenz Hurni

Institute of Cartography and Geoinformation, ETH Zurich – Zurich, Switzerland

Abstract. Nowadays cartography and Geographic Information Systems (GIS) are becoming more and more intertwined with custom software development as for example in Web GIS applications. In this context, this paper raises some concerns that go towards programmer-friendly access of spatial data in the realm of cartography. We argue that there is an object-relational impedance mismatch between the traditional cartographic workflow, where relations are used to store/access spatial data and the object-oriented software development. Therefore, this research promotes integrated cartographic and programmatic application development based upon the concept of using object-orientation for overall handling of geospatial data access and visualization over the Web.

Keywords: databases, object-orientation, Web Cartography

1. Introduction

The development of innovative cartographic and Web-based Geographic Information Systems (Web GIS) applications for visualizing spatial data over the Web often implies custom software implementation. Furthermore, in some cases the efficient and programmer-friendly access and handling of spatial data can become equally important to visualization, mainly due to the importance of software development during the overall implementation of such mapping applications.

We first investigated this issue in (Iosifescu 2009) and then mentioned it in connection with service-driven cartography and service-oriented architectures (SOA) for Web-based mapping applications in (Iosifescu 2011). More recently, we had the opportunity to encounter similar issues in the field of cartography for environmental sciences, more specifically in the frame of

the projects SwissExperiment and OSPER (Open Support Platform for Environmental Research).

The main goal of these environmental projects has been to enable effective real-time environmental monitoring through a common, modern and generic cyber-infrastructure. This goal was effectively implemented through the development and provision of a platform for large-scale sensor network deployment and information retrieval and exploitation (Iosifescu et al. 2010, SwissExperiment 2013).

The SwissExperiment/OSPER Platform is an open support platform for worldwide environmental research experiments, which is designed, implemented and supported by Swiss research institutes (SwissExperiment 2013), at the initiative of the Competence Centre Environment and Sustainability of the ETH Domain (CCES). From the technical point of view, the platform contains mainly three interoperable components: (1) a data documentation system, (2) a management system for sensor data and (3) a service-driven Web-GIS platform. The data documentation system is the SwissExperiment Wiki¹. It is organized as a semantically enhanced metadata store based on the Semantic MediaWiki² and it is being developed by the WSL Institute for Snow and Avalanche Research (SLF). The data management system for sensor data is the Global Sensor Networks (GSN)³. GSN is a Java-based open source middleware designed to facilitate the deployment and programming of sensor networks and it is being implemented by the École Polytechnique Fédérale de Lausanne (EPFL). The third component of the SwissExperiment infrastructure, namely the GIS Platform for Interdisciplinary Environmental Research, serves for the discovery, visualization and download of sensor and spatial data through a service-driven architecture (Iosifescu et al. 2010). This Web-GIS platform is based on GeoVITe⁴ framework and is being developed at ETH Zurich. The GIS Platform for Interdisciplinary Environmental Research is the main cartographic visualization system for the presentation of sensor data (Iosifescu et al. 2010) and recently it uses an additional caching mechanism written in the Java programming language that regularly retrieves and stores the latest sensor measurements in order to improve the performance of the cartographic display.

¹ <http://www.swiss-experiment.ch/index.php/SwissExWiki:Home>, accessed 20 March 2013

² <https://www.semantic-mediawiki.org/>, accessed 20 March 2013

³ <http://sourceforge.net/apps/trac/gsn/>, accessed 20 March 2013

⁴ <https://geodata.ethz.ch/gis/>, accessed 20 March 2013

The SwissExperiment/OSPER platform integrates highly flexible and loosely-coupled components in an interoperable system. As a requirement for interoperability, the sensor measurements are transferred from one component to the other using serialization mechanisms and standardized formats such as web services and GML (Geographic Markup Language). However, in the SwissExperiment/OSPER platform there are three specific use cases where integrated cartographic and programmatic access would make sense: (1) developers would like to integrate the sensor data displayed in the Web GIS directly as objects in an object-oriented programming language such as Java for further processing, (2) the developers would like to save the serialization and deserialization overhead associated with data transfers among the platform components programmed in Java and (3) automatic synchronization between the Java software handling caching of recent sensor data and the relational database used by the Web GIS in order to display this data. These specific use cases and similar use cases as presented in previous work (Iosifescu 2009, Iosifescu 2011), emerge when cartographic presentation in a Web GIS is intertwined with additional data processing implemented in a programming environment.

2. The Object-Relational Impedance Mismatch for Spatial Data

The premise of this research is that current handling of geospatial data based on relational concepts can become cumbersome in the current programming environment where object-orientation is the norm for software development. We will emphasize, for spatial data, the impedance mismatch between existing relational concepts and the object-oriented software development, which is visible in the complicated development process of new mapping applications and the lack of compile-safety when accessing spatial data.

Existing geospatial systems enforces the notion of a layer for geographic data. A layer is a collection of data with a common type or theme as for example cities, rivers, streets or buildings. In the relational model all spatial features of the same category are managed together within the same schema/table, and each record in the table corresponds to a spatial feature.

As a consequence, when working with spatial data objects in an object-oriented programming language such as Java, we encounter two main challenges. First, many conversion steps are necessary for accessing the data and mapping it to objects in a programming language. The second disadvantage is the lack of compile-safety, which may cause errors that are only discovered at run-time. For example, database programming requires pos-

sibly lengthy SQL (Structured Query Language) queries that contain the column names as simple strings. As the compiler is not able to interpret the meaning of these strings, some small typing mistakes may cause errors at run-time.

When considering the conversion steps necessary for accessing the data and mapping it to objects in a programming language, one might argue that there are many third-party libraries that offer useful functionality to alleviate this problem. It is true that the overhead required to read popular GIS data formats and to execute SQL queries from spatial databases is greatly reduced by the use of libraries such as GeoTools⁵. However, when dealing with non-standard geospatial data such as sensor data managed by GSN, the implementation still requires a deserialization of the GML format used for data transfer, or in other words a conversion from the features serialized in GML according to a relational model to generic features stored in a data structure supported by the library. An additional conversion is required to transform these resulting data structures into application-specific spatial objects that can be suitably used to implement the application logic. Moreover, these conversions must take place every time when the software accesses certain data. In computer science terms these kind of conversions that occur repeatedly, are named recurring mappings. *Figure 1* depicts the problem of recurring mappings that usually occur in software dealing with spatial data (Iosifescu 2009).

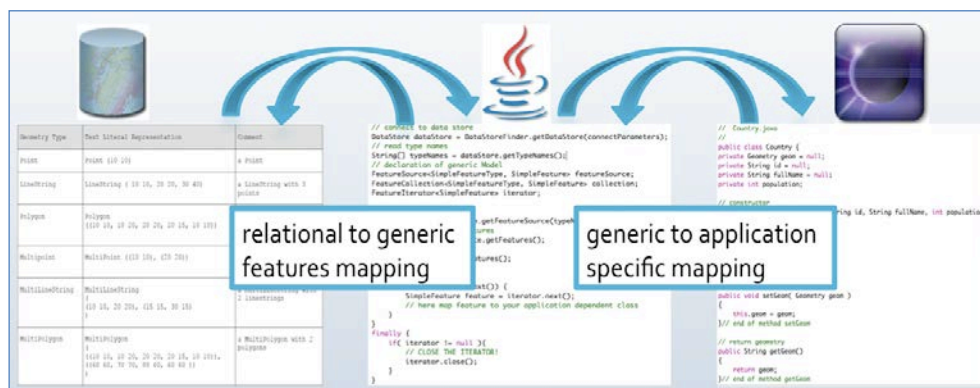


Figure 1. Overview of recurring mappings: first from GML relational features to a data structure accessible from within the code - relational to generic features - and a second one from the resulted data structure to spatial objects that can be suitably used in the application logic - generic to application specific - (Iosifescu 2009).

⁵ <http://geotools.org/>, accessed 20 March 2013

3. Object-Oriented Databases as a Solution to the Object-Relational Impedance Mismatch for Spatial Data in a Web-GIS

On one hand, in order to fulfill its cartographic role a traditional Web-GIS needs access to existing spatial data, which is normally available in relational form. The relational storage of spatial data is a constraint given by the existing GIS and cartographic tools, as well as by standardized formats defined for spatial data exchange (e.g. GML) which expect geographic data to be modeled in this manner. For example, the main cartographic mechanism of the Web-GIS application (which will be presented *Section 4*) and of the GIS Platform for Environmental Research is following a service-driven architecture based on the QGIS mapserver (Iosifescu et al. 2010). QGIS mapserver⁶ is implemented in the C++ programming language and it is able to access popular geographic data sources such as PostGIS⁷ databases, which follow a relational model.

On the other hand, a programmer needs access to objects in the specific programming language being used for the development of application logic. In the specific case of the caching mechanism developed for the GIS Platform for Environmental Research of the SwissExperiment/OSPER project, the application logic was programmed in Java. Therefore, the generic solution was to use an object-oriented database management system (ODBMS) that integrates and builds upon existing spatial database concepts (such as geometry data types, spatial functions, and spatial indexes) in an object-oriented manner. From a technical perspective, for this role we chose the native Java version of the db4objects (db4o) by Versant⁸. It is an open source ODBMS that is marketed as a one-line-of-code database for Java and .NET applications. There are many reasons for choosing db4o beyond its developer-friendliness. Among these, we mention our previous experience in programming with db4o, its ability to work in server mode and its db4o Replication System⁹ (which can be used to automatically replicate objects into a more traditional relational database).

Furthermore, for the transition from a relational to an object-oriented model it is possible to define a mechanism for programmer-friendly devel-

⁶ http://karlinapp.ethz.ch/qgis_wms/index.html, accessed 25 March 2013

⁷ <http://postgis.net/>, accessed 25 March 2013

⁸ <http://www.db4o.com/>, accessed 25 March 2013

⁹ <http://www.db4o.com/about/productinformation/drs/>, accessed 25 March 2013

opment on top of the object-oriented database that can directly transform existing spatial features (which can be available for example in GML or a spatial database) in class definitions that can be stored in the object-oriented database.

In addition, an import module handles the automatic transformation of spatial data into objects and an export module is generating a programmer-friendly Application Programming Interface (API) for accessing spatial data. The input to the import module is a collection of generic spatial features and its output is a collection of spatial objects. The export module is generating a lightweight computation component that provides access, from external code, to the imported spatial objects managed by the ODBMS. The API contains the class definitions as well as the libraries controlling the communication and the transport of spatial objects with the application code.

The final required module required for integrated cartographic and programmatic access to spatial data is a synchronization module with a more traditional spatial Relational Database Management System (RDBMS). Even if the implementation of application logic requires an object-oriented way of dealing with spatial data, we often need to transform spatial objects back to traditional relational databases (such as a PostGIS database) since spatially enabled RDBMS are the norm in the geospatial domain. Therefore, in order to enable interoperability and integration with existing geodatabases and geospatial tools, we need the means to replicate and synchronize an object database (e.g. db4o) with one or more external relational geodatabases (e.g. PostGIS).

As previously mentioned, our Web-GIS application is based on a service-driven cartographic architecture (Iosifescu et al. 2010, Iosifescu 2011). In addition to a Graphical User Interface (GUI), service-driven cartography imposes a three-tier architecture comprising also a spatial database (e.g. PostGIS) and a cartographic Web service (e.g. QGIS mapserver) as an architectural style for the development of Web cartographic applications (Iosifescu 2011).

In the system illustrated in *Figure 2*, while the ODBMS is used for the programmer-friendly access to spatial objects needed during the implementation of application logic, the RDBMS is responsible for serving data for web map publishing. In an attempt to generalize, we believe that a RDBMS is necessary everywhere we need to be compatible with Spatial Data Infrastructures (SDIs), while an ODBMS can act as an additional cache for existing geodatabases and, through the synchronization module, it can be also integrated in wider SDIs.

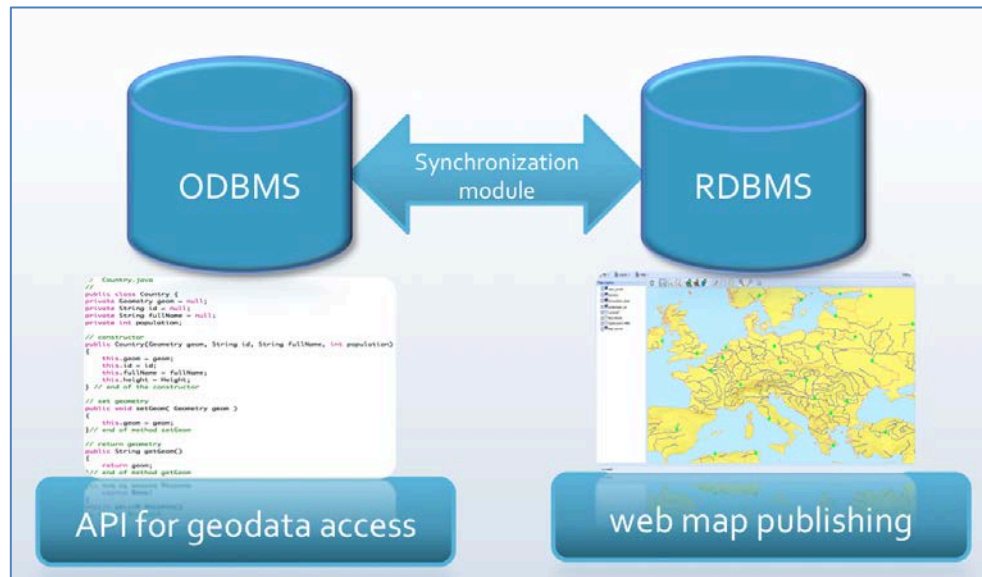


Figure 2. General design for integrated cartographic and programmatic access to spatial data in a Web-GIS (after Iosifescu 2009).

4. Applications of Integrated Cartographic and Programmatic Access to Spatial Data

4.1. Proof-of-Concept Web-GIS

The concept of integrated cartographic and programmatic access to spatial data can be first demonstrated with a generic Web-GIS application (Iosifescu 2009), that allows the users to upload data, change the symbology interactively and finally to access, edit and analyze the spatial data with a programmer-friendly API designed for the Java programming language.

First of all, the proof-of-concept Web-GIS application allows the user to upload data in the Shapefile format. As illustrated in *Figure 3*, the checkbox that enables an integrated cartographic and programmatic access to the uploaded data in the Web-GIS is selected by default. After the upload, the system analyzes the uploaded data, automatically extracts its schema and based on this schema, it generates the corresponding Java classes. These domain classes can be optionally enhanced with methods that define their domain and application specific behavior. Finally, based on the generated Java classes, the system instantiates objects and sets the geometry and the attributes as object properties.

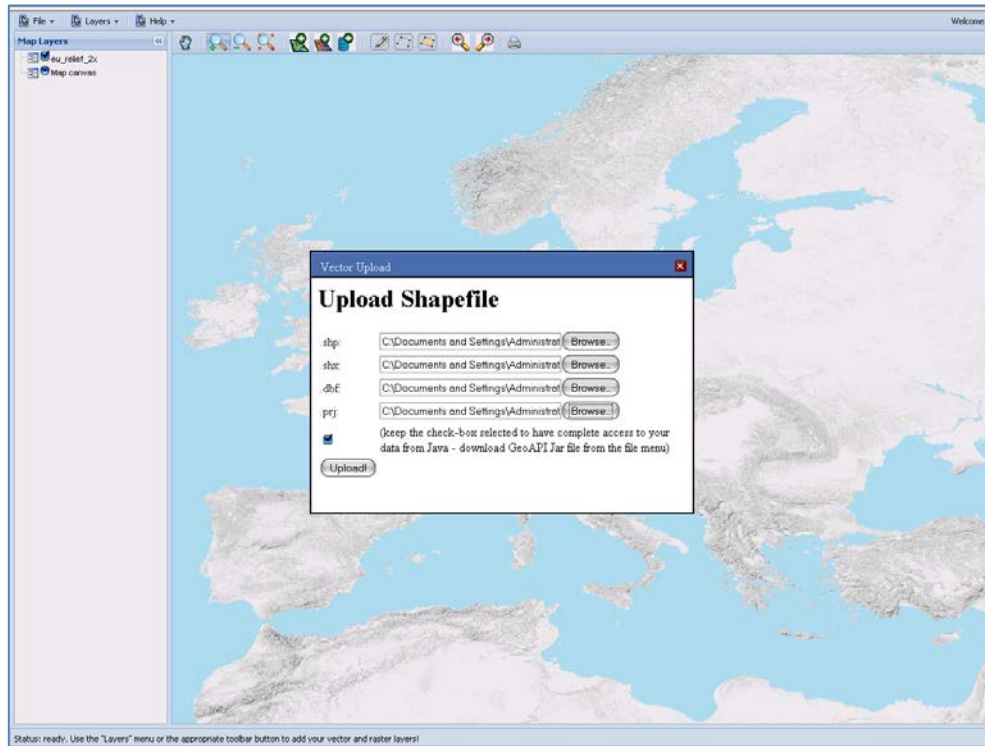


Figure 3. Shapefile upload in the generic Web-GIS for in an integrated cartographic and programmatic access (Iosifescu 2009).

The created objects are then added in an ODBMS, more specifically a db4o instance configured in server-mode. After the objects are inserted in the object-oriented database, a replication mechanism is triggered that synchronizes the db4o object-oriented database with an additional PostGIS relational database.

As explained in *Section 3*, the additional storage of spatial data in a relational spatial database is a constraint given by the existing cartographic tools. Since the generic Web-GIS is following a service-driven architecture (Iosifescu 2011), the PostGIS database is used by the cartographic side of the Web-GIS application in order access this data and create the map that is presented to the user. Furthermore, a user can access basic options for the configuration of symbology as shown in *Figure 4* and as a consequence, the user has the possibility to interactively symbolize the uploaded data sets.

Regarding the cartographic perspective, we can mention that the generic Web-GIS also has additional interactive functionalities such as map export as PDF and an adaptive interface for mobile devices (Iosifescu 2009).

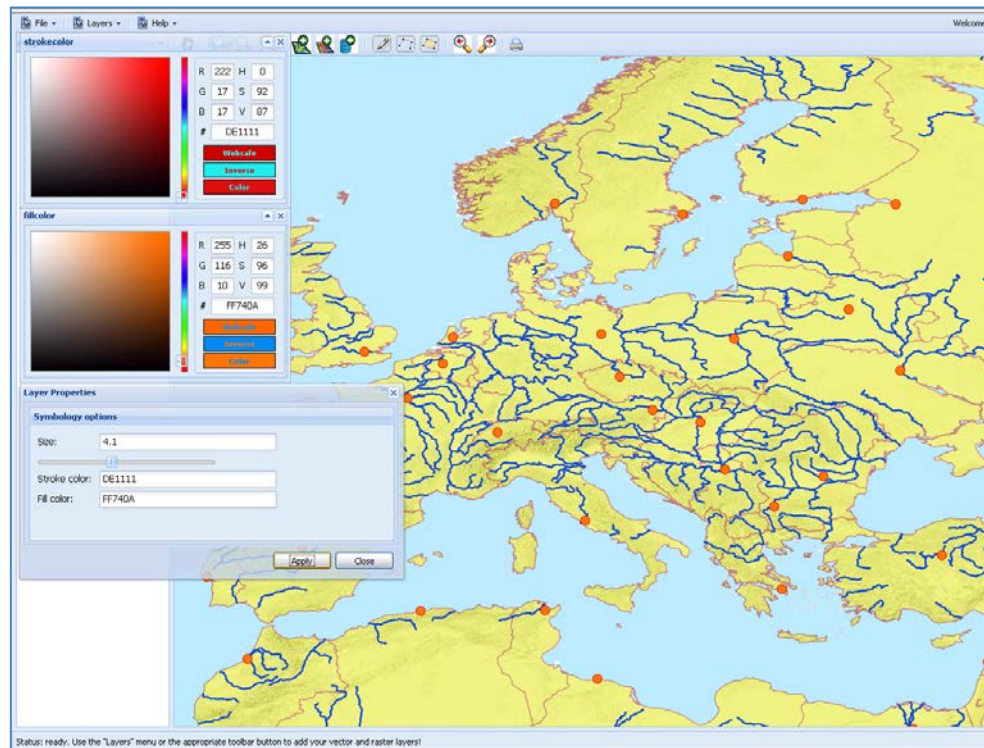


Figure 4. The cartographic side of the generic Web-GIS (Iosifescu 2009).

However, the main advantage of the integrated cartographic and programmatic approach is the efficient access to the data. From the programmatic side of the Web-GIS, developers can efficiently access the data. The access to data implies a simple download of a library (generated dynamically by the Web-GIS for the visualized datasets) and the inclusion of this library in a new Java project. The developer can then use the API implemented by the *GeoObjects* class, shown in *Figure 5*, in order to programmatically access the data directly as objects. The access to spatial data is achievable with a few line of code: first the *GeoObjects* datastore is opened, then a prototype for the class of objects is instantiated and finally a query is performed that returns a collection of spatial objects that match the prototype as illustrated in *Figure 5*. From there on, the developer can use the retrieved objects for implementing the required application logic.

The developer-friendly access to the spatial data uploaded in the generic Web-GIS is enhanced by the location transparency provided by the *GeoObjects* datastore. Therefore, the developers do not even have to configure Internet Protocol (IP) addresses, nor they have to specifically know where the object-oriented database is located, since the included Web-GIS API automatically includes all this information.

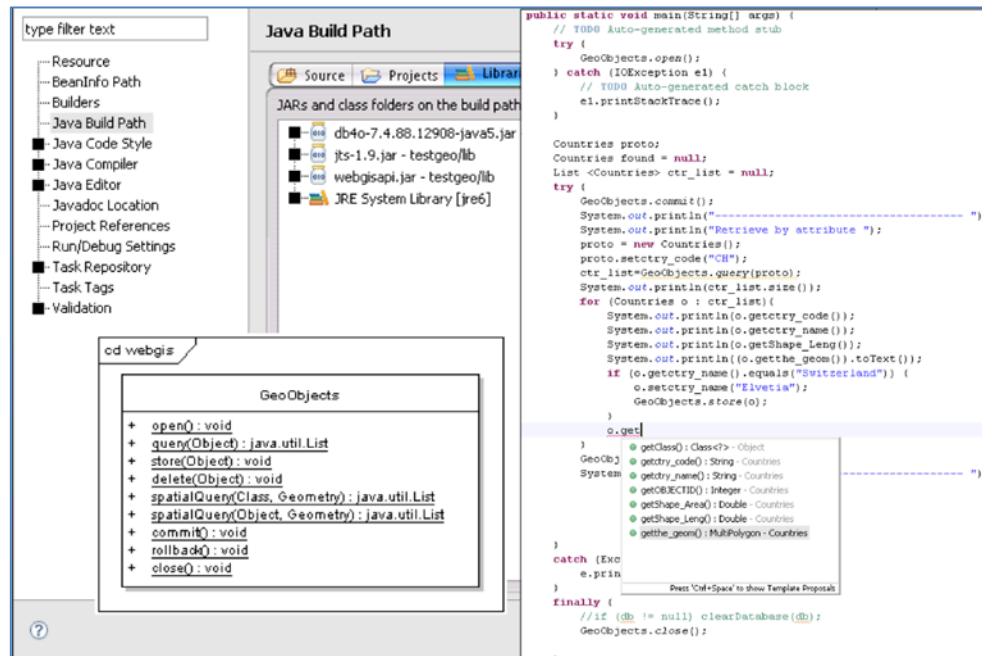


Figure 5. The programmatic side of the generic Web-GIS (Iosifescu 2009).

4.2. Applications in the SwissExperiment/OSPER Project

The object-oriented concepts and design were also applied for the development of the caching mechanism used in the cartographic presentation of sensor data for the SwissExperiment project. A Java servlet regularly connects to the GSN instance that stores sensor data, requests the latest sensor measurements in GML format, converts the received data into various types of sensor objects and then queries the db4o ODBMS for matching objects having similar names.

If the query returns a match, it means that the object (sensor) is already available in the db4o and its properties (sensor measurements) must only be updated with the newest values. If the query does not return a match, it means that we deal with a new sensor and then the new object is simply added to the db4o. Finally, the Java servlet triggers the replication mechanism of the db4o server, which synchronizes the available sensor objects with the corresponding tables in a PostGIS RDBMS. From the PostGIS, the newest sensor data is made available to the GIS Platform for Environmental Research.

In the case of the OSPER project, it is planned to further use the integrated cartographic and programmatic access to the sensor data in the cache in order to implement application logic that will detect sensor measurements anomalies. As soon as these anomalies are detected, they will be displayed in a planned sensor supervision center, which will become available as an additional layer in the GIS Platform. Due to the integrated cartographic and programmatic access to the sensor data implemented in the cache, the GIS platform developers have immediate and full access to sensor objects and can therefore focus on application logic (creating the algorithms necessary to detect sensor measurements anomalies) instead of investing lots of effort for dealing with the basic task of accessing spatial data.

5. Influences of Object-Oriented Concepts in the Wider Context of Web Cartography

The object-oriented concept for accessing spatial data can have additional influences beyond specific cartographic application development. In the wider context of Web Cartography, open standards and web services are currently the newest mechanisms enabling the dissemination of spatial information in real-time over the Web.

In service-driven cartography, Cartographic Web Services (CartoWS) use the OGC standard Symbology Encoding (SE)¹⁰ for styling map data independent of any service interface specification. Furthermore, cartographic extensions for the SE standard allow expressing cartographic rules with advanced point symbolization, patterns for all spatial features, gradients, advanced features filtering and thematic mapping (Iosifescu et al. 2009, Iosifescu 2011).

For fine-grained selection and symbolization of individual geometric features, CartoWS make heavy use of another OGC standard, namely the Filter Encoding Standard (FES)¹¹. For example, based on constraints specified on values of spatial, temporal and scalar properties it is possible to identify a subset of features and render them with a particular predefined symbology.

In this context, a restriction of the relational model is visible in the complicated and lengthy Filter definitions that must be created in order to assign a

¹⁰ <http://www.opengeospatial.org/standards/symbol/>, accessed on 25 March 2013

¹¹ <http://www.opengeospatial.org/standards/filter/>, accessed on 25 March 2013

specific symbology to an individual feature or to a sparse group of features. Filtering works top down, starting with all features in a layer, and gradually eliminating the features that do not fit the defined selection predicates until only the desired feature remains. While this approach is very suitable for the assignment of symbology to entire layers, it is not well suited for individual features because of the significant effort required to define the predicates for the selection.

With an object-oriented approach, filtering and selection may become unnecessary. Individual spatial objects could have, as part of their behavior, one or several symbologies assigned to them, as well as drawing order and possibly additional instructions on how to handle cartographic conflicts. Since the assignment of symbology is made to individual spatial features, the symbolization of entire layers can work bottom up. In this manner, leaving aside any technical considerations regarding rendering performance, spatial objects could be programmed to draw themselves and a map created in this manner could have its content organize itself.

6. Conclusions

Our research identified that developers spend a lot of time with spatial data access instead of programming the application logic and that multiple error prone conversions are unavoidable when transferring data between the runtime domain objects and a relational vector data source. Due to the constraints of the existing relational data model for storing geospatial data, in specific cases developers inadvertently have to deal with the object-relational impedance mismatch, which makes cartographic application development less efficient and more error prone.

With this research we have demonstrated that it is possible to achieve spatial data access and conversions in a completely transparent and object-oriented manner over the Web with an integrated cartographic and programmatic mechanism, which is based on an object-oriented database. The most obvious advantage of an object-oriented spatial data handling is the reduction of the development time, as mappings and transformations are no longer necessary.

Furthermore, the automatic replication of spatial objects to relational spatial databases proves the flexibility and the integration potential with existing cartographic tools and technologies, while making the implementation of custom spatial data processing more programmer-friendly and more integrated with the cartographic visualization.

These integrated cartographic and programmatic functionalities of the Web GIS are possible because, on one hand, the map representation is separated from the GIS data, and on the other hand, the relational spatial data is automatically transformed in Java objects for programmer-friendly development.

Finally, we have presented several ideas about how object-orientation can further influence the cartographic realm and the future integrated map application development for the Internet.

7. Acknowledgements

This work was partially supported by the Competence Center Environment and Sustainability of the ETH Domain (CCES) through the SwissExperiment and OSPER projects. The authors would like to thank all the project partners for their requirements and their interest in cartographic and GIS research.

References

- Alonso G., Casati F., Kuno H. and Machiraju V. (2004) Web Services. Concepts, Architectures and Applications. Springer, Berlin.
- Erl, T. (2008) SOA: Principles of Service Design. Prentice Hall, USA
- Iosifescu I. (2009) Design of a General Architecture for GIS. Master Thesis, ETH Zurich.
- Iosifescu I. (2011) Cartographic Web Services. Doctoral Thesis, ETH Zurich
- Iosifescu I., Hugentobler M., Hurni L. (2009) Web Cartography with Open Standards – A Solution to Cartographic Challenges of Environmental Management. In Environmental Modelling and Software, doi: 10.1016/j.envsoft.2009.10.017.
- Iosifescu I. and Hurni L. (2010) GIS Platform for Interdisciplinary Environmental Research. In Proceedings of the 7th ICA Mountain Cartography Workshop, Borsa, Romania.
- SwissExperiment (2013) The Swiss Experiment Platform. <http://www.swiss-experiment.ch>. Accessed 20 March 2013.